A fast metaheuristic for the travelling salesperson problem with hotel selection

M. Castro*[1], K. Sörensen[1], P. Vansteenwegen[2], and P. Goos[1,2]

[1]University of Antwerp, Belgium
[2]KU Leuven, Belgium

September 3, 2014

Abstract

The travelling salesperson problem with hotel selection (TSPHS) is a recently proposed variant of the travelling salesperson problem (TSP). Currently, the approach that finds the best solutions is a memetic algorithm. However, this approach is unsuitable for applications that require very short computation times. In this paper, a new set-partitioning formulation is presented along with a simple but powerful metaheuristic for the TSPHS. The algorithm is able to obtain very competitive results while remaining at least one order of magnitude faster than the best-performing method so far. The parameters of the metaheuristic were carefully tuned by means of an extensive statistical experiment.

1 Introduction

The *travelling salesperson problem with hotel selection* (TSPHS) (Vansteenwegen et al., 2011) is a recent hierarchical multi-period variant of the TSP (Applegate et al., 2007; Letchford and Lodi, 2007) in which the maximum travel length for each "day trip" is limited, and the salesperson should visit one of the available "hotels" at the end of each day. The objective of the TSPHS is the lexicographical minimisation of the number of day trips and the total travel length. The problem has several practical applications, e.g., the planning of multi-day salesperson tours or the routing of electric vehicles that need to find a recharging station before their battery runs out.

In TSPHS lingo, a "trip" corresponds to a single day of work, i.e., a sequence of visits to customers starting and ending at a hotel, while a "tour" is a set of connected trips that, together, visits all customers. Every trip must start and end in one of the available hotels and should not exceed a given travel length. Of course, the initial hotel of one trip must be the final hotel

*Corresponding author: ✉ Universiteit Antwerpen, Stadscampus B.513, Prinsstraat 13, 2000 Antwerpen, Belgium, ☎ +32 3 265 40 61, 🖷 +32 3 265 49 01, 🖂 marco.castro@uantwerpen.be

of the previous trip. Moreover, the initial and final hotel of the tour, i.e. the initial point of the first trip and the final point of the last trip, are the same and given.

The TSPHS was originally proposed by Vansteenwegen et al. (2011), together with a two-index formulation and an iterated local search algorithm (ILS) (Lourenço et al., 2010) to solve it. In Castro et al. (2013), a more powerful memetic algorithm (MA) (Moscato et al., 2011) which outperforms the heuristic of Vansteenwegen et al. (2011) is presented. However, although the MA is able to find excellent solutions, it is rather complex and can be slow on large instances, i.e., instances with more than 400 customers. For this reason, it is unsuitable for practical situations that require near "real-time" solutions (i.e., computational times of at most a few seconds). In this paper, a fast algorithm is developed that has the advantages of being as simple as possible, competitive in terms of solution quality, and faster by at least one order of magnitude.

Since the TSPHS is a generalisation of the classical TSP, it is also related to other well-known node routing problems that arise in the literature: the multiple travelling salesperson problem (mTSP) (Bektas, 2006), the vehicle routing problem (VRP) (Toth and Vigo, 2002), and the multi-depot vehicle routing problem (MDVRP) (Cordeau et al., 1997; Polacek et al., 2004).

Furthermore, the *hotel selection* requirement is related to problems involving *intermediate facilities* (IF) in which a route is split into trips which start and end at an IF (Angelelli and Speranza, 2002; Kim et al., 2006; Crevier et al., 2007; Tarantilis et al., 2008; Ghiani et al., 2001; Polacek et al., 2008). A detailed description of the TSPHS, including a complete literature review and a mathematical formulation, can be found in Castro et al. (2013).

The rest of the paper is organised as follows: in Section 2, a new formulation for the TSPHS is given. In Section 3, a new metaheuristic is described. In Section 4, a parametric analysis is outlined, while the results obtained by this method are presented in Section 5. Finally, in Section 6, conclusions and avenues for further research are presented.

2 Formulation

Let $\mathcal{H} = \{0,\ldots,s\}$ and $\mathcal{C} = \{s+1,\ldots,s+n\}$ be the set of $s+1$ available hotels and the set of n customers respectively. Using this notation, the TSPHS is defined on the complete undirected graph $G = (\mathcal{V},E)$, where $\mathcal{V} = \mathcal{C} \cup \mathcal{H}$ and $E = \{\{i,j\} | i < j;\ i,j \in \mathcal{V}\}$. A travel time ($c_{ij}$) is assigned to each edge contained in E, while a service time (τ_i) is assigned to every customer $i \in \mathcal{C}$ (with $\tau_i = 0$ for every hotel $i \in \mathcal{H}$).

Let $\mathcal{T}$ be the set of feasible trips, i.e., the trips that start and end at one of the available hotels, and for which the sum of the travel times between the locations (c_{ij}) and the service times (τ_i) at all customers contained in it, does not exceed a maximum time limit L. A trip $t \in \mathcal{T}$ corresponds to an elementary path in G that can be either open, if it starts and ends at different hotels, or closed, if it starts and ends at the same hotel. The subset of closed trips is called $\mathcal{K}$, and the subset of open trips is called $\bar{\mathcal{K}}$.

For each trip $t \in \mathcal{T}$, three parameters, α_{it}, β_{ht} and λ_t, are defined. Parameter α_{it} takes value 1 if trip t visits customer $i \in \mathcal{C}$, and 0 otherwise. Parameter β_{ht} corresponds to the number of times hotel h is contained in trip t. Hence, this parameter might take the values 0, 1 and 2. Parameter λ_t indicates the total length of the trip t, i.e., the length of the elementary path denoted by t (including the service times of the customers).

Finally, let x_t be a binary variable which takes value 1 if trip t is selected and 0 otherwise, and let w_h be an integer variable which denotes the number of times the salesperson arrives at and leaves from hotel h. Using this notation, a set-partitioning formulation for the TSPHS is the following:

$$\min \sum_{t \in \mathcal{T}} (M + \lambda_t)\, x_t \tag{1}$$

$$\text{s. t.} \sum_{t \in \mathcal{T}} \alpha_{it} x_t = 1, \qquad i \in C \tag{2}$$

$$\sum_{t \in \mathcal{K}} \beta_{ht} x_t = 2w_h, \quad h \in \mathcal{H} \tag{3}$$

$$\sum_{t \in \Delta(S)} x_t \leq |\Delta(S)| \left(\sum_{t \in \Psi(S)} x_t \right), S \subseteq \mathcal{H} \setminus \{0\} \tag{4}$$

$$x_t \in \{0,1\},\ w_h \in \mathbb{Z} \tag{5}$$

The objective function (1) minimises the number of trips and the total travel length in lexicographical order. In order to give a higher priority to the minimisation of the number of trips, a big-M approach is used. In this case, a large value M is added to the duration of each of the selected trips so that a solution with fewer trips is always preferred over one with more, regardless of the total travel lengths. Constraints (2) ensure that all customers are visited exactly once, while constraints (3) ensure that whenever the salesperson arrives at a certain hotel, she/he also leaves it. Constraints (4) avoid disconnected cycles, where the set $\Delta(S)$ denotes the set of trips with both endpoints in S, while the set $\Psi(S)$ denotes the set of trips with one endpoint in S and the other endpoint outside S, with $S \subseteq \mathcal{H} \setminus 0$. Constraints (4) are due to the work of Crevier et al. (2007) for the multi-depot vehicle routing problem with inter-depot routes (MDVRPIR).

This proposed formulation differs from the one in Castro et al. (2013) in that it is based on trips instead of arcs. It is therefore suitable for column generation/branch-and-price methods in which the elementary shortest path problem with resource constraints (ESPPRC) (Desaulniers et al., 2005) may be used as sub-problem in order to generate feasible trips (or columns).

It is important to note that the classical TSP is a special case of the TSPHS when $L = \infty$, $\tau_i = 0$ for every $i \in C$, and $s = 0$. Hence, the TSPHS is at least as hard as the TSP, and is therefore also NP-hard. For this reason, it is difficult to optimally solve instances with a moderate or large number of customers. Therefore, a metaheuristic solution is developed in the next section.

3 Solution strategy

In order to solve the TSPHS, a metaheuristic solution strategy is proposed in this section. This strategy combines an order-first split-second method with a simple but powerful heuristic to, respectively, construct an initial solution from scratch and improve that solution. The aim of

this metaheuristic is to be as fast as possible, and to be able to find solutions in sub-second computational times for medium-sized instances, while remaining competitive with the best approach in the literature in terms of solution quality.

In different parts of this solution strategy, several operators are used which can be categorised in the following classes:

(1) two well-known *intra-trip* operators to reduce the trip length: 2-opt (Croes, 1958) and Or-opt (Or, 1976),

(2) two *inter-trip* operators to reallocate a set of customers between two trips: Relocate and Exchange (Laporte et al., 2000),

(3) two *hotel selection* operators (explained below) to either improve the choice of a hotel between two trips (ChangeHotels, see Figure 1) or to remove a hotel between two trips (JoinTrips, see Figure 2), and

(4) two *perturb* operators $P_1(y, \theta_1)$ and $P_2(y, \theta_2)$ to apply perturbations to a given solution y. Operator P_1 randomly relocates $\theta_1\%$ of the customers, while, operator P_2 randomly changes $\theta_2\%$ of the intermediate hotels.

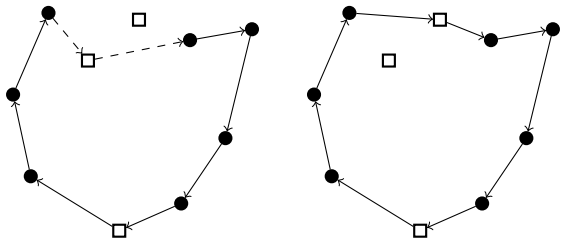
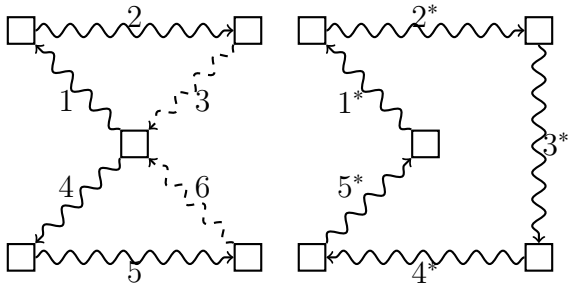


Figure 1: Example of a ChangeHotels move

For operators Or-opt, Relocate and Exchange, strings of up to three consecutive customers are considered to move or exchange.

The aim of the hotel selection operator ChangeHotels is to improve the choice of an intermediate hotel between two consecutive trips in a TSPHS solution. Every intermediate hotel in the solution is tested against the other available hotels. If one or more favourable hotel swaps are identified, the intermediate hotel is replaced with the hotel that leads to the largest improvement in the objective function. In Figure 1, the graph on the left represents a solution before applying the operator, while the graph on the right represents the solution after applying the operator. The white squares indicate the hotels, while the black circles indicate the customers. The hotel between the two dotted arcs in the graph on the left, represents the hotel which is replaced.

The JoinTrips operator attempts to decrease the number of trips by joining any pair of trips which have at least one hotel in common. The operator attempts to remove every hotel in the solution, thereby potentially reversing the direction of one or more trips, while keeping the tour feasible. In the example in Figure 2a, a solution containing six trips is shown. Nodes represent hotels, and arcs represent trips. The labels on the arcs represent the order in which the trips are executed. Trips 1, 3, 4 and 6 have a hotel in common and it is feasible to join



(a) Before (b) After

Figure 2: Example of a JoinTrips move (snaked lines represent trips)

trips 3 and 6. After applying the operator, the resulting solution in Figure 2b contains five trips, where trip 3^* is the outcome of reversing trip 6 and concatenating it with trip 3. Note that the order of executing the trips is also modified in order to reconstruct a feasible tour.

3.1 Construction of an initial solution

In order to construct an initial solution, an order-first split-second method has been implemented. The steps performed during the construction phase are the following:

1. *Order*: Generate a TSP tour by means of the Lin-Kernighan heuristic (Lin and Kernighan, 1973) as implemented by Applegate et al. (2006). This tour starts and ends at the initial hotel (node 0), and visits all customers without considering the time limit L.

2. *Split*: Partition the TSP solution into feasible TSPHS trips using a splitting procedure inspired by that in Prins (2004).

Since the tour constructed in step 1 does not consider the time limit L, it is, usually, infeasible for the TSPHS. The splitting method in step 2 optimally partitions the TSP tour into feasible TSPHS trips in order to construct a feasible solution.

Let S be the sequence of customers in the TSP tour generated during step 1 and let S_i denote the customer at the i-th position of the sequence, where i is indexed from 0 to $n-1$. Using the notation of Prins, the splitting procedure first constructs an auxiliary graph $H = (\mathcal{X}, \mathcal{A}, \mathcal{Z})$ where $\mathcal{X}$ is a set of $(s+1)(n+1)$ vertices, $\mathcal{A}$ is the set of arcs and $\mathcal{Z}$ denotes the weights associated with every arc in $\mathcal{A}$.

The set $\mathcal{X}$ consists of two types of vertices. A vertex $v_i^h \in \mathcal{X}$ with $i < n$ denotes a stay at hotel h before visiting customer S_i. A vertex v_n^h represents a stay at hotel h after visiting all customers.

An arc (v_i^p, v_{j+1}^q) is added to $\mathcal{A}$ if a trip that starts from hotel p, visits all customers from the i-th to the j-th position in the sequence and ends at hotel q, is feasible. The weight $z_{v_k^p v_l^q} \in \mathcal{Z}$ associated with every arc $(v_k^p, v_l^q) \in \mathcal{A}$ represents the length of the trip from v_k^p to v_l^q.

Note that arcs of the form (v_i^p, v_i^q) (with $p \neq q$) are allowed and represent trips that start from hotel p (after visiting customer S_{i-1}) and arrive at hotel q without visiting any customer.

Once the auxiliary graph has been constructed, Dijkstra's algorithm (Dijkstra, 1959) is used to find a shortest path from v_0^0 to v_n^0, i.e., a path starting at hotel 0, visiting all customers, and

ending at hotel 0. Dijkstra's algorithm is a label-setting algorithm for finding shortest paths in directed acyclic graphs that associates a temporary label with each vertex in the graph. This label is an upper bound on the shortest distance from the source vertex to that vertex until the node has been processed. The label becomes permanent when the node has been processed and the label represents the shortest distance instead of an upper bound.

In this splitting procedure, vertices v_0^0 and v_n^0 represent the *source* and *sink* vertices, respectively. Additionally, two distance labels $N_v \in \mathbb{Z}$ and $D_v \in \mathbb{R}$ are associated with each vertex $v \in \mathcal{X}$. N_v and D_v represent the number of trips carried out and the total distance travelled after reaching node v, respectively.

Algorithm 1 Splitting procedure

procedure SPLIT(H, s, t)
  initialise:
    for each vertex $v \in \mathcal{X}$ **do**
      $\pi_v \leftarrow$NIL; $N_v \leftarrow \infty$; $D_v \leftarrow \infty$
    $N_s \leftarrow 0$; $D_s \leftarrow 0$
    $Q \leftarrow \mathcal{X}$; $\overline{Q} \leftarrow \varnothing$
  while $Q \neq \varnothing$
    $u \leftarrow$ select from $Q : (N_u < N_v) \vee (N_u = N_v \wedge D_u \leq D_v) \forall v \in Q \setminus \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for each $v \in \{v|(u,v) \in \mathcal{H}\}$ **do**
      if $(N_v > N_u + 1) \vee ((N_v = N_u + 1) \wedge (D_v > D_u + z_{uv}))$ **then**
        $N_v \leftarrow N_u + 1$
        $D_v \leftarrow D_u + z_{uv}$
        $\pi_v \leftarrow u$
    $\overline{Q} \leftarrow \overline{Q} \cup \{u\}$
    if u = t **then**
      report the predecessor tree given by π
end procedure

The steps performed by the splitting procedure are presented in Algorithm 1. As can be seen, the algorithm keeps a set of nodes to be processed (Q) and a set of nodes already processed ($\overline{Q}$). When the sink node t has been processed, the algorithm terminates. A shortest path with N_t trips and a travelled length of D_t can be obtained by following the predecessor tree from π_t.

This splitting procedure in Algorithm 1, as well as the one of Prins by which this method is inspired, belong to a category of splitting procedures known as order-first split-second methods in the VRP literature. For a recent survey on this kind of methods, the reader is referred to Prins et al. (2014).

3.2 Improvement of a TSPHS solution

After a solution has been constructed, it is subjected to improvement by means of a heuristic described in this section. This heuristic can be seen as an ILS where the typically used local

search is replaced by a variable neighbourhood descend (VND) (Hansen et al., 2008, 2010). This section first describes the VND approach, and then presents the metaheuristic in which it is embedded.

3.2.1 Variable neighbourhood descent

The improvement procedure used in our new algorithm corresponds to a VND. The underlying idea of a VND is that of a systematic change of neighbourhood to seek better solutions. This idea is based on the premise that a local optimum with respect to one neighbourhood is not necessarily a local optimum with respect to another neighbourhood. VND is a deterministic variant of the more general framework called variable neighbourhood search (VNS), which includes a perturbation operator.

Given a solution y, the VND tries to minimise the function

$$F(y) = \sum_{x_t \in y} (M + \lambda_t)\, x_t + \omega \sum_{x_t \in y} \max(\lambda_t x_t - L, 0),$$

where L is the maximum trip length. In the VND, a solution may be infeasible with respect to the trip length constraint and, hence, the length λ_t of a trip t may be larger than L. This infeasibility, if any, is given a penalty proportional to a large constant ω. In this way, the VND attempts to recover from infeasibilities.

The VND comprises four neighbourhoods ($N_k, k = 1, \ldots, 4$) defined by the *inter-trip* and *hotel selection* operators, namely Relocate, Exchange, ChangeHotels and JoinTrips. These neighbourhoods are sequentially explored, in the order mentioned. In the pseudo-code shown in Algorithm 2, the structure of the VND is presented.

Algorithm 2 Variable neighbourhood descent

```
procedure VND(y)
    k ← 1
    while k ≤ 4 do
        y′ ← Search(N_k, y)
        if y′ is better than y then
            y ← y′
            k ← 1
        else
            k ← k + 1
    report y
end procedure
```

Furthermore, as can be seen in Algorithm 3, the search over each neighbourhood is performed in a best improvement fashion, i.e., the complete set of candidate solutions within a neighbourhood is explored and the best one is selected. If a better solution is found, the new solution is improved by means of the *intra-trip* operators (2-opt and Or-opt). Also, the search over each neighbourhood is repeated as long as a better solution can be found in it.

Algorithm 3 Search over a neighbourhood N_k

procedure SEARCH(N_k, y)
  $\hat{y} \leftarrow y$
  repeat
    $\hat{y}' \leftarrow \arg\min_{\breve{y} \in N_k(\hat{y})} F(\breve{y})$
    if $\hat{y}'$ is better than $\hat{y}$ **then**
      $\hat{y} \leftarrow$ Improve $\hat{y}'$ with 2-OPT/OR-OPT
  until no improvement has been found
  report $\hat{y}$
end procedure

3.2.2 Proposed metaheuristic P-LS

This section outlines the complete metaheuristic, henceforth labelled P-LS. The metaheuristic operates in the domain of solutions reachable by the VND, but, in order to introduce diversification into the search, two perturbation operators, P_1 and P_2, are used. The metaheuristic has been designed to iteratively perturb and improve a given solution in a way that is similar to an ILS.

Algorithm 4 P-LS metaheuristic

require
  Number of iterations $i_{\max}$
  Parameters θ_1 and θ_2
begin
  $T \leftarrow$ Tour produced by the Lin-Kernighan heuristic
  $H \leftarrow$ Construct auxiliary graph from T
  $y \leftarrow$ SPLIT(H, v_0^0, v_n^0)
  for $i = 1$ **to** $i_{\max}$ **do**
    $k \leftarrow 1$
    while $k \leq 2$ **do**
      $y' \leftarrow P_k(y, \theta_k)$
      $y'' \leftarrow \text{VND}(y')$
      if y'' is better than y **then**
        $y \leftarrow y''$
        $k \leftarrow 1$
      else
        $k \leftarrow k + 1$
  report y
end

In Algorithm 4, the pseudo-code of the complete metaheuristic is shown, where y is the initial solution constructed with the order-first split-second method described in Section 3.1,

$i_{\max}$ is the number of iterations, and θ_1 and θ_2 are the parameters of the perturbation operators P_1 and P_2, respectively.

As can be seen, at each iteration, the algorithm executes the sequence perturb-and-improve. First, the perturbation operator P_1 is applied. That operator is applied repeatedly for as long as it leads to an improved solution. When no improvement can be found any more, perturbation operator P_2 is activated. If that operator leads to an improved solution,then the search switches back to operator P_1. Otherwise, the current iteration ends.

4 Parametric analysis

In this section, a statistically designed experiment is presented, the aim of which is to determine the best parameter configuration for the algorithm developed in Section 3.

Since the construction method for the initial solution is deterministic, the performance of the heuristic strategy depends only on the parameters of the P-LS identified in Algorithm 4, namely, $i_{\max}$, θ_1 and θ_2.

The experiment has been conducted on a new set of randomly generated TSPHS instances, which have been designed to contain small, medium and large numbers of customers, as well as diverse numbers of available hotels. These new instances are generated in such a way that the optimal solution is known. The procedure used to generate these instances is the same as the one used in Vansteenwegen et al. (2011) and works as follows.

In order to create a single instance: (1) generate a set of random points, (2) solve the classical TSP to optimality by using the Concorde TSP solver (Applegate et al., 2006), and (3) given a certain number of trips, insert artificial hotels along the optimal TSP tour to produce TSPHS instances in such a way that the optimal tour length is the same for both problems.

In Table 1, the parameters which have been used to generate the random instances are shown. For each combination, five different instances have been generated, resulting in a total of 125 instances. Table 2 provides an overview of the values that were tested for the algorithm's three parameters.

Parameter	Value
Number of customers	75, 100, 200, 300, 500
Number of trips	5, 10, 15, 20, 25

Table 1: Parameters of the random instances

Parameter	Values	Num. levels
$i_{\max}$	5, 10, 15, 20	4
θ_1	0.05, 0.10 ,0.20, 0.30 0.4, 0.5, 0.6, 0.7, 0.8	9
θ_2	0.05, 0.10 ,0.20, 0.30 0.4, 0.5, 0.6, 0.7, 0.8	9

Table 2: Parameters of the algorithm and the values tested in the experiment

In order to measure the influence of the three parameters on the quality of the solutions obtained by the algorithm, a full factorial experiment has been performed, and the influence of the parameters on two performance measures has been analysed: the objective function value ($F(y)$) and the total CPU time. Furthermore, in order to minimise the occurrence of infeasible solutions, a large penalty ω was used in the objective function, $F(y)$. More specifically, the penalty ω was set to 10000.

For each of the two performance measures, a type III analysis of variance (ANOVA) model has been estimated, involving a random effect for each instance and fixed main effects and two-way interaction effects of the parameters.

Table 3 displays the results for the two estimated ANOVA models. In Column 1, the parameter or interaction is shown, while, in Columns 2 and 3, the p-values are presented for the objective function value model and for the CPU time model, respectively. Only p-values of significant effects are shown.

Source	p-values	
	F	Time
$i_{\max}$	< 0.0001	< 0.0001
θ_1	< 0.0001	< 0.0001
θ_2	< 0.0001	< 0.0001
$i_{\max} \times \theta_1$		< 0.0001
$i_{\max} \times \theta_2$		< 0.0001
$\theta_1 \times \theta_2$	0.0200	< 0.0001

Table 3: Significant effects in ANOVA models for both performance measures

From Table 3, it is clear that all three parameters have a significant effect on both performance measures. For the CPU time consumption, all possible 2-interactions are significant, while for the objective function value, only the interaction between θ_1 and θ_2 is significant. In Figures 3 and 4, the relevant mean plots are shown.

Figure 3 shows that the more time is spent solving this problem, the better the solution obtained. This is a typical behaviour of well-designed metaheuristics. Furthermore, as can be expected, the CPU time increases linearly with the number of iterations.

The effects of parameters θ_1 and θ_2 as well as their interaction with parameter $i_{\max}$ is similar with respect to the CPU time consumption. The larger the level of the perturbation, the larger the computational time. This is explained by the fact that the perturbation operators attempt to diversify the search by creating different and usually inferior solutions. Generally, the more different a solution is from a locally optimal solution, the worse it is. Therefore, a larger extent of perturbation requires an additional improvement effort in subsequent iterations.

Figure 4, visualises the interaction effect of θ_1 and θ_2 on F. The figure shows that θ_1 should be set to a value between 0.20 and 0.40, while θ_2 should have a small value, between 0.05 and 0.20.

Based on the results of the statistically designed experiment, the parameter levels displayed in Table 4 were selected for the P-LS algorithm. In the next section, the results obtained by the metaheuristic on four sets of benchmark instances are presented. All the results reported use the settings in Table 4.

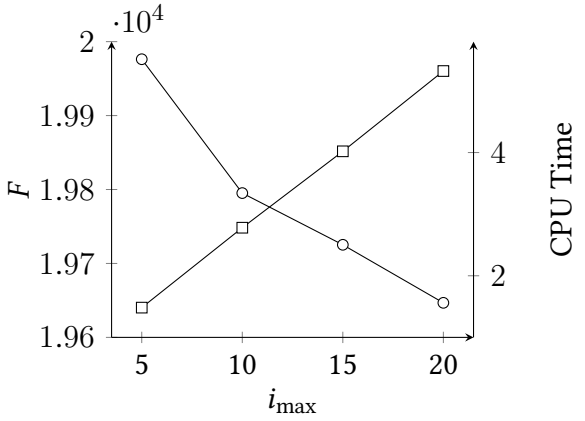


Figure 3: Influence of parameter $i_{\max}$ on the objective function value (circle) and on the computational time (square)

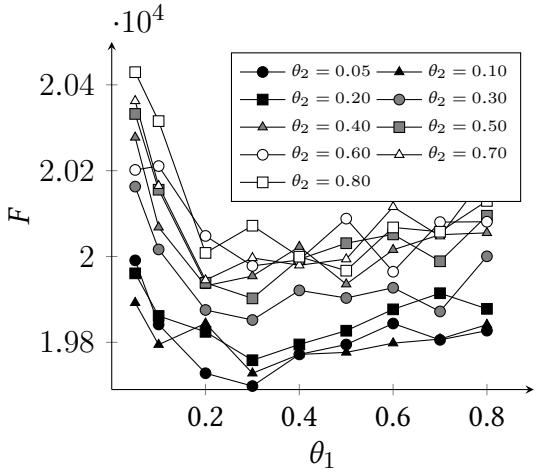


Figure 4: Interaction $\theta_1 \times \theta_2$ vs F

Parameter	Value
$i_{\max}$	20
θ_1	0.30
θ_2	0.10

Table 4: Selected settings of the three parameters of the P-LS

5 Results

In this section, the results produced by the P-LS algorithm for each of four sets of benchmark instances are presented. The first set (SET 1) is created from benchmarks for the capacitated vehicle routing problem with time windows (CVRPTW) from Solomon (1987). The second set (SET 2) contains four subsets made up from SET 1 by including only the first 10, 15, 30 and 40 customers. The third set (SET 3) is created from benchmarks for the classical TSP and for which the optimal solution is known. Three different subsets are generated, containing three, five and 10 hotels. Finally, the fourth set (SET 4) is created from the same benchmarks for the TSP, by imposing an arbitrary time limit L and including randomly generated hotels. Unlike for SET 3, for SET 4 no optimal solutions are known. All instances are publicly available at `http://antor.ua.ac.be/tsphs`.

Due to the fact that the MA developed in Castro et al. (2013) clearly outperforms the heuristic of Vansteenwegen et al. (2011) (referred to as I2LS), only detailed results from the MA and the P-LS are shown. However, for every set of instances, a table summarising the results is also presented. That summary table does include the I2LS heuristic.

All experiments were run on an Intel Core i7 850 processor with 2.93 GHz and 4 GB of RAM.

For SET 1, containing 16 instances, the results are presented in Table 5. The first two columns contain the name and the number of customers of each instance. The next columns show the number of trips, the total travelled length and the computational time, both for the MA and the P-LS algorithm. Since the results produced by the MA are taken as reference, an additional column showing the percentage gap between both approaches is included for the P-LS algorithm:

$$\text{Gap} = 100 \times \frac{\text{Length(P-LS)} - \text{Length}(MA)}{\text{Length}(MA)}.$$

As can be seen, the gap formula only takes into account the travelled length (including the service time) and not the number of trips. The reason to do this is because the number of trips found by both the MA and the P-LS algorithm is the same in almost every case.

Table 5 shows that, for instances pr01, pr04 and pr07, the P-LS was able to find the best-known solution, while, for the rest of the instances, the gap with respect to the best-known solutions was never larger than 1.15%. For all instances, the same number of trips was obtained.

Furthermore, from the summary shown in Table 6, it can be seen that the P-LS algorithm produced an average gap of only 0.33% in a CPU time which is two orders of magnitude smaller than that of the MA, and similar to that of the I2LS. The percentage gap of the I2LS is, however, large, with an average of 2.64%.

For SET 2, which is made up of four subsets containing 13 instances each, the results are presented in Tables 7, 8, 9 and 10 (for 10, 15, 30 and 40 customers, respectively). The values under the columns labelled "Best" correspond to optimal solutions in case these are available (see Castro et al. (2013)), or to best-known solutions (produced by the MA) in case the optimal solution is unknown. An asterisk indicates instances for which no optimal solution is available.

For all instances with 10 and 15 customers, the P-LS algorithm was able to find all optimal solutions. For the subsets of 13 instances with 30 and 40 customers, the P-LS algorithm was

		MA			P-LS			
Instance	N	Trips	Length	Time (s)	Trips	Length	Time (s)	Gap(%)
c101	100	9	9595.6	24.1	9	9596.9	0.1	0.01
r101	100	8	1704.6	24.0	8	1717.4	0.2	0.75
rc101	100	8	1674.1	29.5	8	1674.3	0.2	0.01
c201	100	3	9560.0	16.2	3	9563.1	0.1	0.03
r201	100	2	1643.4	11.6	2	1648.1	0.1	0.28
rc201	100	2	1642.7	12.4	2	1644.3	0.2	0.09
pr01*	48	2	1412.2	2.8	2	1412.2	0.0	0.00
pr02	96	3	2543.3	18.1	3	2551.3	0.2	0.31
pr03	144	4	3415.1	48.4	4	3421.1	0.3	0.17
pr04*	192	5	4217.4	165.8	5	4217.4	0.6	0.00
pr05	240	5	4958.7	331.8	6	4974.7	1.1	0.32
pr06	288	7	5963.1	327.7	7	6032.0	1.6	1.15
pr07*	72	3	2070.3	13.1	3	2070.3	0.0	0.00
pr08	144	4	3372.0	64.9	4	3399.9	0.4	0.82
pr09	216	5	4420.3	228.0	5	4445.7	1.1	0.57
pr10	288	7	5940.5	409.0	7	5991.5	2.4	0.85
Avg.				108.0			0.5	0.33

Table 5: Results for SET 1. An asterisk indicates a solution for which the P-LS algorithm was able to find the same solution as the MA

	MA	I2LS	P-LS
Num. best-known	16/16	0/16	3/16
Min. gap (%)	0.00	0.41	0.00
Max. gap (%)	0.00	5.36	1.15
Avg. gap (%)	0.00	2.64	0.33
Avg. time (s)	108.0	1.9	0.5

Table 6: Summary for SET 1

Name	Best		P-LS		
	Trips	Length	Trips	Length	Gap(%)
c101	2	1452.2	2	1452.2	0.00
r101	2	379.8	2	379.8	0.00
rc101	2	303.2	2	303.2	0.00
pr01	1	590.4	1	590.4	0.00
pr02	1	745.6	1	745.6	0.00
pr03	1	632.9	1	632.9	0.00
pr04	1	683.4	1	683.4	0.00
pr05	1	621.2	1	621.2	0.00
pr06	1	685.2	1	685.2	0.00
pr07	1	795.3	1	795.3	0.00
pr08	1	707.2	1	707.2	0.00
pr09	1	771.7	1	771.7	0.00
pr10	1	611.9	1	611.9	0.00
Avg.					0.00

Table 7: Results for instances in SET 2 containing 10 customers.

Name	Best		P-LS		
	Trips	Length	Trips	Length	Gap(%)
c101	2	1452.2	2	1452.2	0.00
r101	2	379.8	2	379.8	0.00
rc101	2	303.2	2	303.2	0.00
pr01	1	590.4	1	590.4	0.00
pr02	1	745.6	1	745.6	0.00
pr03	1	632.9	1	632.9	0.00
pr04	1	683.4	1	683.4	0.00
pr05	1	621.2	1	621.2	0.00
pr06	1	685.2	1	685.2	0.00
pr07	1	795.3	1	795.3	0.00
pr08	1	707.2	1	707.2	0.00
pr09	1	771.7	1	771.7	0.00
pr10	1	611.9	1	611.9	0.00
Avg.					0.00

Table 8: Results for instances in SET 2 containing 15 customers.

Name	Best		P-LS		
	Trips	Length	Trips	Length	Gap(%)
c101*	3	2863.2	3	2863.6	0.01
r101	3	655.2	3	655.2	0.00
*rc101**	*3*	*705.5*	*4*	*683.8*	*-3.08*
pr01	1	964.8	1	964.8	0.00
pr02	2	1078.3	2	1078.3	0.00
pr03	1	952.5	1	952.5	0.00
pr04	2	1091.6	2	1091.6	0.00
pr05	1	924.7	1	924.7	0.00
pr06	2	1063.2	2	1063.2	0.00
pr07	2	1130.4	2	1130.4	0.00
pr08	2	1006.2	2	1006.2	0.00
pr09	2	1091.4	2	1091.4	0.00
pr10	1	918.9	1	918.9	0.00
Avg.					-0.24

Table 9: Results for instances in SET 2 containing 30 customers. An asterisk indicates instances for which no optimal solution is available.

Name	Best		P-LS		
	Trips	Length	Trips	Length	Gap(%)
c101*	4	3866.1	4	3867.3	0.03
r101*	4	862.8	4	873.5	1.24
*rc101**	*4*	*850.3*	*5*	*870.8*	*2.41*
pr01	2	1160.5	2	1160.5	0.00
pr02	2	1336.9	2	1336.9	0.00
pr03	2	1303.4	2	1303.4	0.00
pr04	2	1259.5	2	1259.5	0.00
pr05	2	1200.7	2	1200.7	0.00
pr06	2	1242.9	2	1242.9	0.00
pr07	2	1407.0	2	1410.3	0.23
pr08	2	1222.2	2	1222.2	0.00
pr09	2	1284.2	2	1284.4	0.01
pr10	2	1200.4	2	1200.4	0.00
Avg.					0.30

Table 10: Results for instances in SET 2 containing 40 customers. An asterisk indicates instances for which no optimal solution is available

able to find 11 and 5 of the best-known solutions, respectively. For all instances, the gaps are smaller than 2.5%.

A special remark has to be made concerning instance rc101 in the subset with 30 customers for which a gap of -3.08% with respect to the best-known solution is reported. This is due to the fact that the P-LS algorithm was able to find a solution with a shorter length than the MA, but the MA solution involves a smaller number of trips. Every time the number of trips is different for the two solution approaches, these results are indicated in italic in our tables with results.

	I2LS				P-LS			
Num. customers	10	15	30	40	10	15	30	40
Num. best-known	5/13	5/13	1/13	1/13	13/13	13/13	11/13	5/13
Min. gap (%)	0.03	0.25	0.48	0.10	0.00	0.00	-3.08	0.01
Max. gap (%)	7.28	6.31	5.88	6.40	0.00	0.00	0.01	2.41
Avg. gap (%)	2.03	1.26	3.10	3.13	0.00	0.00	-0.24	0.30

Table 11: Summary for SET 2

In Table 11, it is possible to see that, for every group of instances contained in SET 2, the I2LS produced gaps higher than 5%, while, the P-LS obtained average gaps of at most 0.3% and was able to find a larger number of best-known solutions. As mentioned, the "best" solution corresponds to the optimal solution when it is available, and to the solution value produced by the MA when it is not. For this reason, solutions produced by the MA are assumed to have gaps of 0% and the MA is not included in Table 11.

The results for SET 3, containing 48 instances divided in three groups of 16 instances, are shown in Tables 12, 13 and 14. The three groups of instances differ in the number of extra hotels used when generating them. This set was generated in such a way that near-optimal (and

in most cases optimal) solutions are available.[1] For both the MA and the P-LS, the columns display, for each instance, the number of trips of the solution, the travelled length, the CPU time and the gap with the best-known solution.

Name_*N*	TSP	MA				P-LS			
		Trips	Length	Time (s)	Gap (%)	Trips	Length	Time (s)	Gap (%)
eil_51	426	4	426	3.8	0.00	4	426	0.0	0.00
berlin_52	7542	4	7542	3.2	0.00	4	7542	0.0	0.00
st_70	675	4	675	7.0	0.00	4	675	0.0	0.00
eil_76	*538*	*4*	*538*	*27.7*	*0.00*	*5*	*556*	*0.0*	*3.34*
pr_76	108159	4	108159	14.6	0.00	4	108159	0.1	0.00
kroa_100	21282	4	21282	14.2	0.00	4	21282	0.1	0.00
kroc_100	20749	4	20749	15.1	0.00	4	20749	0.0	0.00
krod_100	21294	4	21294	15.9	0.00	4	21294	0.1	0.00
rd_100	7910	4	7910	15.7	0.00	4	7910	0.1	0.00
eil_101	629	4	629	16.9	0.00	4	629	0.1	0.00
lin_105	14379	4	14379	16.4	0.00	4	14379	0.1	0.00
ch_150	6528	4	6528	35.7	0.00	4	6528	0.2	0.00
tsp_225	3916	4	3916	93.4	0.00	4	3916	0.5	0.00
a_280	2579	5	2591	228.4	0.46	5	2615	0.8	1.39
pcb_442	*50778*	*4*	*50778*	*672.1*	*0.00*	*5*	*51144*	*3.7*	*0.72*
pr_1002	259045	4	259045	3172.8	0.00	4	259045	34.5	0.00
Avg.				272.1	0.02			2.5	0.34

Table 12: Results for SET 3 with 3 extra hotels

Table 15 shows the summary of the results for the 48 instances in SET 3. It is clear that the P-LS is able to produce very competitive results. Both the MA as well as the P-LS algorithm are able to keep their average gaps below 1%, but it is the MA which is able to find the largest number of the best-known solutions. The most striking result is that the P-LS algorithm results in average gaps of 0.34%, 0.53% and 0.39% in computational times that are two orders of magnitude smaller than the computational times required by the other two approaches. It is also clear that the P-LS outperforms the I2LS in terms of solution quality for this set.

For SET 4, containing 15 instances, the detailed results are presented in Table 16. For this set, no optimal solutions are known. Hence, the results produced by the MA are taken as benchmark. The column "Gap (%)" displays the percentage gap between both approaches.

Table 17 summarises the results of applying the three methods to SET 4. The MA is able to find the largest number of best-known solutions. However, the P-LS algorithm was able to find new best solutions for four instances contained in this set, namely, eil_76, krod_100, tsp_225 and pr_1002. Despite the fact that the gap for instance kroa_100 is negative, it does not improve the solution found by the MA, since the number of trips for the solution found by the P-LS is larger.

Like for the instances in SET 3, the P-LS algorithm is much faster than the other two heuristics for instances in SET 4. It produces an average gap of 0.25%, while the average gap for the I2LS approach amounts to 10.05%.

[1] In Table 14, the gap value is omitted for instance berlin_52. The reason for this is that the MA was able to find a solution with one trip less than the solution with optimal TSP length, which contains nine trips.

Name_*N*	TSP	MA				P-LS			
		Trips	Length	Time (s)	Gap (%)	Trips	Length	Time (s)	Gap (%)
eil_51	426	6	426	3.5	0.00	6	426	0.0	0.00
berlin_52	7542	6	7542	3.6	0.00	6	7542	0.0	0.00
st_70	675	6	675	7.1	0.00	6	675	0.0	0.00
eil_76	538	6	538	9.3	0.00	6	566	0.1	5.20
pr_76	108159	6	108159	8.1	0.00	6	108159	0.1	0.00
kroa_100	21282	6	21282	14.5	0.00	6	21282	0.1	0.00
kroc_100	20749	6	20749	13.8	0.00	6	20749	0.1	0.00
krod_100	21294	6	21294	14.7	0.00	6	21294	0.1	0.00
rd_100	7910	6	7910	13.5	0.00	6	7910	0.1	0.00
eil_101	629	6	629	16.3	0.00	6	629	0.1	0.00
lin_105	14379	6	14379	15.4	0.00	6	14379	0.1	0.00
ch_150	6528	6	6528	40.2	0.00	6	6528	0.2	0.00
tsp_225	3916	6	3916	86.8	0.00	6	3916	0.4	0.00
a_280	2579	7	2646	193.1	2.59	7	2652	0.7	2.83
pcb_442	*50778*	*6*	*50778*	*483.2*	*0.00*	*7*	*51087*	*3.1*	*0.60*
pr_1002	*259045*	*7*	*259774*	*3882.4*	*0.28*	*6*	*259045*	*20.5*	*0.00*
Avg.				300.3	0.18			1.6	0.53

Table 13: Results for SET 3 with 5 extra hotels

Name_*N*	TSP	MA				P-LS			
		Trips	Length	Time (s)	Gap (%)	Trips	Length	Time (s)	Gap (%)
eil_51	426	10	426	3.3	0.00	10	426	0.0	0.00
berlin_52	7542	8	7864	3.9	-	9	7542	0.0	0.00
st_70	675	10	675	6.6	0.00	10	675	0.0	0.00
eil_76	*538*	*11*	*538*	*9.0*	*0.00*	*12*	*567*	*0.1*	*5.39*
pr_76	108159	11	108159	7.9	0.00	11	108159	0.1	0.00
kroa_100	21282	11	21282	14.1	0.00	11	21282	0.1	0.00
kroc_100	20749	11	20749	13.8	0.00	11	20749	0.1	0.00
krod_100	21294	11	21294	14.1	0.00	11	21294	0.1	0.00
rd_100	7910	10	7910	14.3	0.00	10	7910	0.1	0.00
eil_101	629	11	629	15.2	0.00	11	629	0.1	0.00
lin_105	14379	10	14379	15.3	0.00	10	14379	0.1	0.00
ch_150	6528	11	6528	35.8	0.00	11	6528	0.2	0.00
tsp_225	3916	11	3916	79.9	0.00	11	3916	0.4	0.00
a_280	*2579*	*11*	*2613*	*134.1*	*1.31*	*12*	*2596*	*0.7*	*0.65*
pcb_442	*50778*	*11*	*51774*	*511.4*	*1.96*	*12*	*50919*	*2.6*	*0.27*
pr_1002	259045	11	259045	3224.1	0.00	11	259045	11.9	0.00
Avg.				256.4	0.21			1.0	0.39

Table 14: Results for SET 3 with 10 extra hotels

	MA			I2LS			P-LS		
Num. extra hotels	3	5	10	3	5	10	3	5	10
Num. best-known	15/16	14/16	13/16	0/16	0/16	1/16	13/16	13/16	13/16
Min. gap (%)	0.46	0.28	1.31	2.86	6.39	6.67	0.72	0.60	0.27
Max. gap (%)	0.46	2.59	1.96	17.37	22.36	20.47	3.34	5.20	5.39
Avg. gap (%)	0.02	0.18	0.21	12.73	13.82	11.08	0.34	0.53	0.39
Avg. time (s)	272.1	300.3	256.4	460.2	459.9	775.6	2.5	1.6	1.0

Table 15: Summary for SET 3

Name_*N*	MA			P-LS			
	Trips	Length	Time (s)	Trips	Length	Time (s)	Gap (%)
eil_51	6	429	3.8	6	436	0.0	1.63
berlin_52	7	8642	4.2	7	8642	0.0	0.00
st_70	6	723	8.5	6	731	0.0	1.10
eil_76	6	548	12.4	6	539	0.0	-1.65
pr_76	*6*	*118061*	*8.2*	*7*	*118719*	*0.1*	*0.55*
kroa_100	*6*	*22343*	*19.2*	*7*	*22044*	*0.1*	*-1.34*
kroc_100	6	20933	12.8	6	21116	0.1	0.87
krod_100	6	21664	17.3	6	21464	0.1	-0.93
rd_100	*6*	*8244*	*24.2*	*7*	*8245*	*0.1*	*0.01*
eil_101	6	634	22.8	6	652	0.1	2.83
ch_150	6	6647	54.7	6	6728	0.2	1.21
tsp_225	6	4571	118.7	6	4502	0.9	-1.51
a_280	6	2646	158.7	6	2658	0.9	0.45
pcb_442	6	54339	872.5	6	55134	5.7	1.46
pr_1002	7	292690	3423.4	7	290110	29.5	-0.89
Avg.			317.4			2.5	0.25

Table 16: Results for SET 4

	MA	I2LS	P-LS
Num. best-known	10/15	0	6/15
Num. new best-known	-	-	4
Min. gap (%)	-	2.09	-1.65
Max. gap (%)	-	18.03	2.83
Avg. gap (%)	-	10.05	0.25
Avg. time (s)	317.4	310.7	2.5

Table 17: Summary for SET 4 instances

6 Conclusions

The travelling salesperson problem with hotel selection (TSPHS) is a relatively new variant of the TSP that has several practical applications. Two metaheuristic methods exist in the literature, the iterated local search approach (I2LS) of Vansteenwegen et al. (2011) and the memetic algorithm (MA) of Castro et al. (2013), which is the method that produces the best solutions in terms of quality. However, for applications requiring solutions in short computational times (e.g., real-time applications), the MA is too slow and the I2LS does not generate high-quality solutions. In this paper, a simple but powerful heuristic solution method, named P-LS, has been presented for the TSPHS. The new approach combines an order-first split-second construction method with a fast improvement phase. By intensively exploiting problem-specific hotel selection operators, and optimising the parameters of the P-LS using a rigorous statistically designed experiment, this approach is competitive in terms of solution quality when compared to the best method in the literature, while at the same time being at least one order of magnitude faster.

Future work can focus on extensions of the TSPHS, involving, for example, time windows, multiple salespeople and hotel costs.

Acknowledgements

The first, second and fourth author gratefully thank the Fonds voor Wetenschappelijke Onderzoek - Vlaanderen (FWO) as well as the COMEX Project (Combinatorial optimization: metaheuristics & exact methods) for financial support.

References

E. Angelelli and M. G. Speranza. The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, 137(2):233–247, 2002.

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde TSP solver. `http://www.tsp.gatech.edu/concorde`, 2006.

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2007.

T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

M. Castro, K. Sörensen, P. Vansteenwegen, and P. Goos. A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research*, 40(7):1716–1728, 2013.

J. F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

B. Crevier, J. F. Cordeau, and G. Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756–773, 2007.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6): 791–812, 1958.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*. Springer, 2005.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

G. Ghiani, G. Improta, and G. Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37(3):134–143, 2001.

P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno-Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, 2008.

P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno-Pérez. Variable neighborhood search. In *Handbook of Metaheuristics*, pages 61–86. Springer, 2010.

B. I. Kim, S. Kim, and S. Sahoo. Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33(12):3624–3642, 2006.

G. Laporte, M. Gendreau, J. Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4-5):285–300, 2000.

A. N. Letchford and A. Lodi. The traveling salesman problem: a book review. *4OR*, 5(4): 315–317, 2007.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

P. Moscato, R. Berretta, and C. Cotta. Memetic algorithms. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, 2011.

I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University, 1976.

M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6): 613–627, 2004.

M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5): 405–423, 2008.

C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

C. Prins, P. Lacomme, and C. Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

C. D. Tarantilis, E. E. Zachariadis, and C. T. Kiranoudis. A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. *INFORMS Journal on Computing*, 20(1):154–168, 2008.

P. Toth and D. Vigo, editors. *The vehicle routing problem*, volume 9 of *Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, 2002.

P. Vansteenwegen, W. Souffriau, and K. Sörensen. The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society*, 63(2):207–217, 2011.